CALIBRATION OF NEURAL NETWORKS USING GENETIC ALGORITHMS, WITH APPLICATION TO OPTIMAL PATH PLANNING

Terence R. Smith

Department of Computer Science
University of California at Santa Barbara
Santa Barbara, CA 93106

Gilbert A. Pitney

Department of Computer Science
University of California at Santa Barbara
Santa Barbara, CA 93016

Daniel Greenwood

Netrologic
4241 Jutland
San Diego, CA 92117

ABSTRACT

Genetic algorithms (GA's) are used to search the synaptic weight space of artificial neural systems (ANS) for weight vectors that optimize some network performance function. GA's do not suffer from some of the architectural constraints involved with other techniques and it is straightforward to incorporate terms into the performance function concerning the metastructure of the ANS. Hence GA's offer a remarkably general approach to calibrating ANS. GA's are applied to the problem of calibrating an ANS that finds optimal paths over a given surface. This problem involves training an ANS on a relatively small set of paths and then examining whether the calibrated ANS is able to find good paths between arbitrary start and end points on the surface.

1. INTRODUCTION AND PROBLEM STATEMENT

Massively parallel computing devices composed of many elementary processing elements (PE's), connected in a simple and local manner, offer the possibility of computing complex input-output relationships relatively quickly. One approach to achieving massive parallelism involves the use of many identical and simple processing elements (PEs) and relatively local communication links between PEs. The artificial neural system (ANS) represents one class of such systems that are currently the object of much investigation. Each PE of an ANS produces as its output a single, bounded, real-valued number. An ANS is a collection of PEs, each of which takes as input the weighted outputs of other PEs. The ANS architectures considered in this paper consist of networks of synchronous, binary threshold units (BTU's, see Egecioglu, Smith and Moody, 1987).

The behaviour of the ANS (i.e., the mapping it is able to compute) is largely determined by the set of weights by which the output of a given PE is multiplied before being taken as input to another PE. A major problem in ANS design involves the determination of an appropriate set of connection weights between the PE's for computing a given mapping.

In this paper, our primary focus of attention concerns an essentially unexplored technique for programming ANS's, namely genetic algorithms (GA's). A secondary focus of attention concerns the construction of an ANS that is able to compute "good" paths over some surface, using GA's and a set of input-output exemplars to program the system. In particular, we are interested in the ability of this programming technique to construct an ANS that significantly generalizes over the set of input-output pairs.

1.1. Research Reported in this Paper

The research reported in this paper is of an exploratory and empirical nature, since the behaviour of both ANS's and GA's are currently difficult to analyse in a formal manner. Our basic approach to the problem involves:

a) construction of a surface over which "good" paths are to be computed and computation of globally optimal paths between all given pairs of points on the surface (using the Dijkstra algorithm) to produce a training set of input-output patterns (start-end points, optimal paths)

b) establishment of a priori constraints on the architecture of the ANS

c) choice of which variant of GA to employ in calibrating the ANS

d) a set of training runs in which a subset of input-output patterns are used to program the connection weights

e) tests of the ANS on the remaining input-output patterns to determine how well the GA performs in generalizing over its training set.

The main purpose of the experiments reported here was to provide intuition into the application of GA's for calibrating ANS's, particularly in relation to the path planning problem. More systematic investigations of the problem are now in progress.

2. NEURAL NETWORK PROGRAMMING METHODS

Two main problems in ANS design are 1) finding a suitable network architecture, in terms of connection topology and type of PE's, and 2) determining the weights of the ANS. To date, no automatic procedure for designing a network architecture for a given input/output behaviour exists, although as discussed later, GA's may be applied to solve this problem (we apply intuition as a guide to designing the architecture). The main purpose of this paper is to propose a relatively new solution to the second problem.

We define the programming of an ANS as adjusting the weights such that the network can compute a desired input to output mapping. There are numerous techniques for programming an ANS, many of which are best suited for particular problem domains, or limited to specific network architectures. These techniques may be currently classified into two groups, the first of which is based on finding connection weights in terms of predetermined functions of the problem parameters and the second of which is based on some form of search over the space of weights.

The first class includes associative memory techniques (Hopfield, 1982) and techniques based on finding quadratic forms that express a problem in terms of a set of constraints (Hopfield and Tank, 1985).

Concerning the second class, one may classify the techniques according to the degree of "localness" of the search procedure. Most learning procedures perform a search over the weight space to minimize some performance criterion of the network. The search techniques include gradient descent, gradient descent with annealing, and guided random search. Examples of such techniques include, respectively, back propagation (Rumelhardt and McClelland, 1986); the master/slave formalism (see Lapedes and Farber, 1986); and guided accelerated random search (GARS, Mucciardi, 1972).

In this paper, we propose the genetic algorithm as a neural network programming procedure.

2.1. Genetic Algorithms

The GA can be viewed as a relatively global search procedure based upon population genetics (Holland, 1975). We apply the GA as a function optimizer to the weight space of an ANS to maximize some performance function of the network. The major strengths of the GA as a function optimizer are its ability to search efficiently and effectively high dimensional, multimodal, noisy, and discontinuous surfaces. Since the GA is being used purely to search the weight space, there are no restrictions on network architecture. There are also no restrictions on the terms of the performance function.

The basic GA maintains a population of individuals. In the case of the function optimization problem, each individual represents a point in the parameter space of the performance function, and is represented by a binary string encoding of the parameter vector. Each individual is evaluated, and a new generation is produced by selecting individuals on the basis of their performances for reproduction. Because higher performing individuals are selected more often for reproduction, and due to the recombination effects of the crossover operator, there is a pressure towards higher performing individuals being accepted into the population.

The basic algorithm is:

1. Randomly generate a population, P_0, of N members. Set t=0.
2. For all i=1..N, compute and save the performance measure $\mu(P_i^t)$.
3. If converged, then STOP. Best individual of last population is solution.
4. Compute selection probabilities
$$\rho_i^t = \mu(P_i^t) / \sum_{j=1}^{j=N} \mu(P_j^t)$$
5. Generate next generation, P^{t+1}, by choosing individuals via selection probabilities for reproduction using genetic operators. Set t=t+1. Goto 2.

The genetic operators used in this paper are crossover and mutation. Crossover recombines two parent vectors to produce an offspring vector by concatenating the segment to the left of a random crossover point in the first parent with the segment to the right of the same crossover point in the second parent. The mutation operator, with a low probability, alters bits in the offspring. The combined effect of crossover, mutation and selection allows genetic algorithms to search very high dimensional spaces efficiently.

One of the most challenging problems in ANS learning procedure design is the assignment of credit to processing elements which are responsible for a system's high performance, especially when those elements are only active early in a long chain of actions which eventually leads to reward from the environment. The GA solves the credit assignment problem by selection. Individuals which contain good weight vectors are rewarded by a higher probability of recombination and reproduction. Thus the weights are held accountable for network performance.

3. THE APPLICABILITY OF THE ANS AND ITS ASSOCIATED PROGRAMMING METHODS TO THE PATH PLANNING PROBLEM

A secondary goal of this investigation is to program an ANS in such a manner that it contains an efficient, internal representation of a "cost" surface characterized in terms of some set of efficient paths over the surface. This representation should permit the network to compute a "good" path between two arbitrary points on the surface, given only those two points. Since only a subset of the precomputed optimal paths over the surface are presented to the network during the learning phase, the network must be able to generalize.

In most of the work to date on the programming of ANS's to compute specific functions, researchers have employed the stable states of the ANS as a basis for representation. For any ANS, there is a fixed number of such states. Hence the ability of an ANS to compute a given function is ultimately limited by this capacity constraint. However, different approaches to representing a given computation may result in more or less efficient ANS. Hence part of our research has concentrated on different approaches to network representations and their relative efficiencies.

4. A PRELIMINARY INVESTIGATION OF GA'S FOR PROGRAMMING ANS'S TO SOLVE THE PATH PLANNING PROBLEM

GA's may be used to modify the synaptic weights of the ANS in order to maximize the net's performance in finding optimal paths. The resulting network ideally accepts an input pattern representing start and end positions on a given surface, and produces an output pattern representing a least cost path from the given start and end points.

4.1. A Priori Hypotheses Concerning the Topology of the Connection Weights

As noted above, the topology of the connection weights may be an important factor in determining the efficiency of a network with a given number of PE's. Hence we explored four alternative topologies, while keeping the number of "hidden" PE's constant at 20.

A quad tree structure was suggested by prior experience with computational architectures for solving path planning problems (Smith and Parker, 1987). This architecture embodies the hypothesis that the pertinent features of the landscape required for the ANS to predict optimal paths can be best represented in a hierarchical fashion, with the more abstract, higher order features of the surface encoded at the top of the hierarchy. It presumes that computation proceeds from the top downward, with higher levels guiding (constraining) the computation at lower levels of the tree. We examined three such architectures:

a) FFQ is a feed-forward quad tree structure with 4 layers (see Figure 1)

b) RQ (see Figure 2) is a modification of FFQ with recurrent connections between layers

c) RQNNN is the same as RQ, except for the addition of next nearest neighbor connections between units on a layer (see Figure 5)

d) FIH (see Figure 3) involves 20 fully-connected hidden units.

Quad tree topology is shown in Figure 4.

4.2. The Landscape

The surface investigated is derived from a topological map of a 40 square kilometer area of the Sierra Madre Mountains in California mapped onto an 8x8 square grid of pixels. Each pixel is represented as a node in a four-connected transition cost graph, in which each link represents a bidirectional, symmetric cost. The derived cost graph is then used as input to Dijkstra's algorithm to compute optimal paths, and to the GA's objective function in order to gauge the performance of each ANS.

4.3. Objective Functions

The GA uses an objective function to evaluate the performance of each member of the population. In this case, the individual is an ANS, and the task is to predict the optimal path over a surface between two points. After presentation of the input pattern to the ANS, and after the network relaxes, the objective function computes an error measure between the optimal path predicted by the network and the true optimal path. The network predicts a path by turning on those neurons in the output layer which correspond to nodes in the transition graph, which in turn correspond to points on the surface.

It is helpful to choose a performance measure which facilitates the genetic search. The objective function is a mapping from the weight space of the ANS to a single performance value. As a general rule, the objective function should posses some degree of 'smoothness' in the region about the solution point in the weight space. This means that any change in the weights in the direction of the optimum should yield a higher performance value. For a discussion of how various types of objective function surfaces affect search procedures, see Ackley (1987).

Two basic performance functions are used in these simulations. The first function, P1, incorporates three terms: 1) an incorrect link cost, 2) an incorrect pixel cost, and 3) convergence time, as defined below:

$$x = \sum_j [abs(l_j^{opt} - l_j^{net})(l_j^{opt} cost(l_j^{opt}) + l_j^{net} cost(l_j^{net}))] + 20\sum_i [abs(i^{opt} - s_i)] + T$$

$$P_1 = 100/(1+x) \qquad (4)$$

Where, all references to links and neurons refer to the output layer, and

l^{opt} = { 1 if link is between adjacent neurons on the optimal path; 0 otherwise}.
l^{net} = { 1 if link is between adjacent neurons turned on by the network; 0 otherwise}.
cost(l) = cost of traversing link l.
i^{opt} = { 1 if the ith pixel lies on the optimal path; 0 otherwise}.
s_i = state of neuron i.
T = relaxation time of the network.
j = index over all links in output layer.

The incorrect link cost term penalizes the network for predicting paths which either 1) contain a linkage between two adjacent neurons which does not exist in the optimal path, or 2) lack a linkage which does occur in the optimal path. The amount of penalization is just the sum of the costs of traversing such linkages. The incorrect pixel count term penalizes the network for predicted paths which either 1) contain points which do not appear in the optimal path, or 2) lack points which do appear in the optimal path. The amount of penalization is proportional to the number of such incorrect pixels. The third term is the number of time constants the net takes to relax. The network is said to have relaxed when the activity pattern of the output layer has remained constant for seven time constants.

The second performance function, P2, is designed to overcome the apparent deficiencies in P1, and also to stress to the network the importance of well-formed paths. Note that the second term in P1 enforces the constraint that pixels predicted by the network lie on the optimal path. However, it will also penalize a

predicted path of near optimal cost if that path does not geographically coincide with the optimal path. This, to some degree, violates the smoothness criterion for good objective functions. Thus, the second term in P1 is replaced by two terms which impose the constraint that the output pixels be on a well formed path, not necessarily the optimal. The first term is modified to some degree, violates the smoothness criterion for good objective functions. Thus, the second term in P1 is replaced by two terms which impose the constraint that the output pixels be on a well formed path, not necessarily the optimal. The first term is modified slightly from P1, but still enforces the optimal path constraint:

$$x = 20[\sum_k [s_k(1 - ncount(k))^2] + \sum_h [s_h(2 - ncount(h))^2]] + abs(C - \sum_j cost(l_j^{net})) + T$$

$$P_2 = 100/(1+x) \qquad (5)$$

Where

ncount(i) = number of neighboring neurons of i which are on.
C = the cost of traversing the optimal path.
k = index over both path endpoints.
h = index over all other points.

Note that in P_2 the terms enforcing well-formed paths are weighted most heavily.

4.4. Representation of the Weight Vector

Generally, the ordering of gene values in the GA control string can strongly affect convergence, especially in the absence of an inversion operator. The control string is a binary string encoding of the weight vector. Each weight is encoded in 8 bits in two's complement binary, and ranges in value from -128 to 127. The weights are then concatenated to make up the control string.

Two ordering schemes are used. In the first, called LR, the weights ordered from left to right correspond to a top down ordering in the network. For example, weights on connections to the 2x2 hidden layer in the FFQ network are encoded at the leftmost end of the control string. The second scheme, called Q, distributes the weights over the control string in quad tree order. Thus, weights on connections to the top of the network hierarchy are not grouped together, but are distributed throughout the corresponding sectors over the length of the binary string.

In the GA used in this study, crossover is the main operator for generating new weight vectors for evaluation. Since it is assumed that the abstract features of the landscape allowing the ANS to generalize will be encoded in the hidden unit weights, it is expected that encoding scheme Q will facilitate search more than scheme LR by allowing crossover to generate offspring with a greater variety of hidden weights. Scheme Q can only be applied to the quad tree networks.

4.5. Reproductive Plan 4 (R4)

The variant of GA used in these simulations is the elitist expected value model (Reproductive Plan - R4) discussed in De Jong (1975). Two genetic operators are used in this model: mutation and crossover. An elitist model transfers the best performing individual of the current population intact into the next generation. This policy slightly favours local search, and is found to speed convergence. The expected value model drastically reduces stochastic errors by replacing the use of the random variable in the selection process by a counting scheme based upon the expected value of the selection probability. This prevents any statistical fluctuation which might, for example, cause a high-performing member of the population to be overlooked during reproduction.

4.6. Training Methods

Two different training modes were used during the programming of the network. In the first, T1, the same training set of data is shown to the networks over all generations. In T2, at each generation the networks are evaluated on unique and disjoint subsets of the training set. Thus, in training mode T2, the total number of data points in the training set is the product of the constant size of the training subset per generation times the number of generations.

5. SIMULATIONS AND RESULTS

Table I summarizes the GA parameters used in the simulations. The parameters of the nine separate experiments are summarized in table II. The experimental procedure used to train and evaluate each network is discussed below.

5.1. Experimental Procedure

First, the training and test data sets were prepared. Dijkstra's algorithm was applied to the cost graph representing the landscape to find the optimal path between all pairs of points on the surface. Each data point is a tuple consisting of an input pattern encoding the start and end points, and an output pattern encoding the optimal path from the start to the end point. Subsets of the data were allocated to a training set and a test set.

The experiment proceeded in two phases. During the learning phase, the training set data was used by the GA's performance function to search the weight space of a particular ANS. After learning, the capability of the network to generalize was measured on the test data set.

5.1.1. Learning Phase

Before any particular run, the network architecture was specified. An initial population of weight vectors was randomly generated. Each member was evaluated by simulating the equations governing the corresponding ANS and measuring its performance in a series of trials in which the net attempts to complete the correct output pattern for a given corresponding input pattern.

After the net relaxed, or when the maximum time allot-

ted for the network to relax was exceeded (50 cycles in our simulations), the objective function was applied to the output layer. The average over all trials was taken as the performance of that particular net. This performance value was then used by the GA to assign selection probabilities to individuals for reproduction. The learning phase ends when the genetic search converges.

5.1.2. Test Phase

In the test phase, the best performing member of the last generation was evaluated on the test data set. To allow comparisons, the performance function used in the test phase, P3, was the same for all runs:

$$x = \sum_k [s_k(1 - ncount(k))^2] + \sum_h [s_h(2 - ncount(h))^2] + abs(C - \sum_j cost(l_j^{net}))$$

$$P_3 = 100 - x/2 \qquad (6)$$

P3 is a variant of P2, with equal emphasis on cost and path terms, without the convergence time term, and linear in x.

5.2. Description of Results.

The results of the nine runs are displayed in tables III, IV, and V.

Table V shows the number of generations each run took to converge, as well as the total number of paths in the training set. When training method T1 was used, the performance vs. generations curve was monotonically non-decreasing, and the GA was considered to converge when no improvement had been made in the performance value of the best network over 12 generations. With training method T2, since the training data was different for each generation, the performance vs. generations curve was not monotonic, and the GA was considered converged when no improvement could be seen in the average performance of the best network over about 20 generations. Run 6 took a long time to converge, and was stopped at 288 generations. It should be noted that run 6 converged with respect to performance on long length paths. Before it was stopped, it was continuing to increase in the performance on short to medium length paths.

Table III shows the value of the performance (P3) of the best network of the last generation of each run on the training data. Entries marked by a dash in the table signify that no paths of length indicated by the column heading were contained in the training set. Performance values are sorted by path length, and are averaged over the number of paths of that length in the training set, which varied from run to run. A performance of 100 is a perfect score, indicating that the network correctly predicted the optimal path.

The main results of this work are given in table IV. The performance (P3) is calculated as in table III, but using the data in the test set, and averaged over a constant number of trials per path length, as indicated in the last row of the table.

5.3. Discussion of Results

Although none of the ANS's did a perfect job of consistently predicting optimal paths during the test phase, we gained some insight into the problems of training a network to generalize, of weight vector representation in the GA and of network architectures for this problem.

5.3.1. Performance on the Training Data

Referring to table III, performances using the training data in runs 1, 3, 4, and 5 show the ability of the best network to correctly predict the single training path of length 15 pixels. Only one pixel was missing from the middle of the predicted path in run 3, giving that network a suboptimal score of 98. Run 2 shows the performance on 5 training paths of lengths ranging from 11 to 15 points. In this run, one path was predicted correctly, one had a few extra pixels in the output layer turned on, and 3 paths were halfway complete.

The networks of runs 1 through 3 did not use their hidden units in predicting the optimal path. Only when the architecture was changed from FFQ to RQ and FIH in runs 4 and 5, respectively, that is, when bottom up connections were added, did any hidden units come into play.

Runs 6 through 9 were trained using one or five different paths per generation. This training method always caused the best network to use its hiddens in the computation, and led to better generalization capabilities on the test data. Because of the training method used, the total training set sizes in runs 6 through 9 were much larger, as shown in table V. There are no significant differences between the performances of the nets in runs 6 to 9 on the training and test data. These networks made greater use of their hidden units, and learned early in the training phase to generalize. This was expected, since the T2 training method does not allow any one path to be seen by a network for more than one generation, thus discouraging the 'memorization' of a specific pattern.

5.3.2. Performance on the Test Data

The measures of performance of the best networks on the test data give some indication of their ability to generalize (see table IV).

5.3.2.1. Best Networks

The network which had the most consistently high average performance over all path lengths was that of run 8 (RQ,P2,T2,5,Q). Given any two points as input, the network often made a reasonable approximation to a path between them, keeping disconnected pixels to a minimum. Short to medium length paths would sometimes complete correctly, but the network had trouble with longer paths.

The next best network, in terms of generalization capability, was that of run 6 (FIH,P1,T2,1,LR), which performed very well on short to medium length paths, but did much more poorly on longer paths.

Though the network of run 5 (FIH,P1,T1,1,LR) shows high performance values for medium to long length paths, its ability to generalize was nil. No matter what the input pattern to this network, the output pattern would usually be the same path used in the training phase. P3 would score the pattern highest for long optimal path lengths because the path was connected, and usually had a traversal cost similar to the optimal path of the given inputs. The only cost penalty was in the lack of connections to the true path endpoints, which is small.

5.3.2.2. Comparisons Between Runs

Despite the paucity of runs, a comparison of simulation results in table IV suggests some interesting, though inconclusive, results.

Comparing runs 1 through 5 with runs 6 through 9 indicates that the training method T2, i.e., showing each generation a different training set, is sufficient to produce generalization capability in the networks. It is not known whether T1, with a much larger training set size, would also induce generalization.

A comparison of the results of runs 1 and 3 show that the quad tree ordering of weights in the genetic control string gave a slight improvement in performance over a simple top down encoding.

Comparing runs 1 and 4 suggests that bottom up connections increased performance.

Run 7 was somewhat of an anomaly, in that it should not have differed much from run 6. We believe that, for the number of alleles in the chromosome, the population size was too small, and the GA had insufficient gene variability to sustain a global search, and thus found a local minimum.

Comparing runs 8 and 9 indicate that adding next nearest neighbor connections within layers actually decreased generalization capability. In fact, a consideration of the qualitative observations on the data of runs 1, 4 and 5, which were all trained with T1 and one training pattern, shows that the addition of feedback connections widens the basin of attraction about the single learned memory vector. Thus, the networks with more feedback connections, and trained under T1, more often produced the same training pattern as output independent of the input pattern.

6. CONCLUSIONS

We have proposed a general technique for programming ANS's using GA's. Unlike most techniques, the GA imposes no constraints on network architecture or performance function. As a result, novel terms, relating not only to the network's performance in the particular task environment, but also to meta variables of the network, may be incorporated. For example, our objective functions P1 and P2 included the network convergence time as a term to be minimized. In run 5, on the training set, the GA found a weight vector capable of perfectly predicting the optimal path after 73 generations, with a network convergence time of 14 time constants. After 20 more generations, the GA had decreased the convergence time to 12, then finally to 10 time constants. Such a criterion as speed of computation would be difficult to incorporate into most other network programming procedures.

Concerning the solution to the optimal path planning problem, it is apparent to us that 20 hidden BTU's is insufficient to solve the problem as posed. A major limitation is the number of stable states that are feasible using an ANS with only 20 hidden units. The path planning problem is hard for the network to solve because of the minimum of input information, and because it convolves two problems, namely finding well-formed paths and finding optimal paths. The fact that the GA could not find a weight vector to solve the problem was because of architectural constraints. We doubt that any other learning procedure could have solved this problem with the given number and type of neurons.

Even though the networks were not able to always predict optimal paths, the simulations showed us the importance of knowledge guided search through the experimental parameter space.

This work suggests two directions for future research. First, for the short term, a more rigorous experimental approach is needed to explore network architectures for solving the path planning problem. It appears that a hierarchical network architecture, with bottom up feedback, is the most promising structure. The number of hidden units and the power of the PE's should also be increased.

The second and more fundamental area of research involves the first problem of ANS design: finding a suitable network architecture for a particular problem. This includes number and type of PE's, and connectivity. Here especially the GA appears to be a natural candidate solution, because of its role in the evolution of the human nervous system.

The solution would involve finding a good encoding of an ANS architecture in terms of a representation suitable for manipulation by the GA, and a developmental plan to translate that encoded representation (genotype) into the corresponding network (phenotype). Such a plan may be a set of growth rules, of the type discussed in Lindenmayer (1976) or proposed recently by Wilson (1987). The performance of each network can then be evaluated in the given task environment. Furthermore, if the network is able to learn during its evaluation phase, that is, if the connection strengths are not solely determined by evolution, Hinton and Nowlan (1987) argue that the learning capability would provide an easier 'evolutionary path' toward the optimal network architecture.

7. REFERENCES

Ackley, D.H., "Stochastic Iterated Genetic Hill-Climbing," Doctoral Dissertation, Carnegie-Mellon University, Pittsburgh, PA, 1987

De Jong, K., "Analysis of the Behavior of a Class of Genetic Adaptive Systems," Ph.D. thesis, Dept. Comp. and Com. Sciences, Univ. of Michigan, 1975.

Egecioglu, O., Smith, T. R., and Moody, J., "Computable Functions and Complexity in Neural Nets," in J. Casti

(Ed.), Proceedings of the Abisko Conference on Neural Computation, Elsevier (in press), 1987.

Hinton, G., and Nowlan, S.,"How Learning Can Guide Evolution," Complex Systems, Vol. 1, No. 3, June 1987, pp. 495-502.

Holland, J., *Adaptation in Natural and Artificial Systems*, Univ. of Michigan, Ann Arbor, 1975.

Hopfield, J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proc. Natl. Acad. Sci. USA, Vol. 79, 1982, pp. 2554-2558.

Hopfield, J. and Tank, D., "'Neural' Computation of Decisions in Optimization Problems," Biological Cybernetics,52,141, 1985.

Lapedes, A., and Farber, R., "Programming a Massively Parallel, Computation Universal System: Static Behavior," Conf. Proc. of Neural Networks for Computing, Snowbird, UT, 1986, pp. 283-298.

Lindenmayer, A., and Rozenberg, G. (eds.), *Automata, Languages, Development*, Amsterdam: North-Holland, 1976.

Mucciardi, A.N., "Neuromime Nets as the Basis for the Predictive Component of Robot Brains," *Cybernetics, Artificial Intelligence, and Ecology*, H.W. Robinson and D. E Knight(Ed.s), Spartan Books, Bensalem, PA., 1972, pp. 159-193.

Rummelhart, D., and McClelland, J., *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*. MIT Press, 1986, Chapter 8.

Smith, T. R. and Parker, R. E., "An Analysis of the Efficacy and Efficiency of Hierarchical Procedures for Computing Trajectories over Complex Surfaces," European Journal of Operations Research, 30,1987, pp. 327-338.

Wilson, S., "The Genetic Algorithm and Biological Development," Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms, July 28-31, 1987, MIT, Cambridge, MA. pp. 247-251.

8. FIGURES

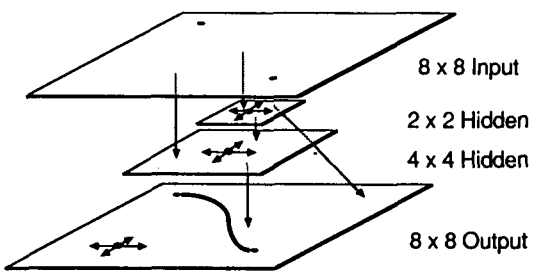

Figure 1 - Feed Forward Quad Architecture

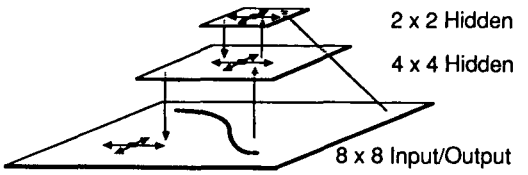

Figure 2 - Recurrent Quad Architecture

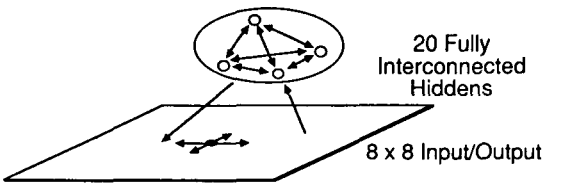

Figure 3 - Fully Interconnected Hiddens

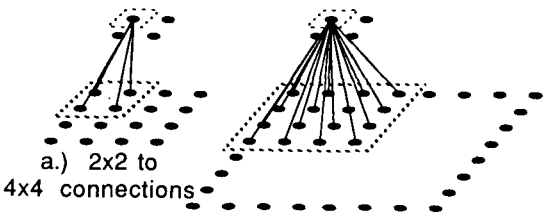

Figure 4 - Quad Tree Topology

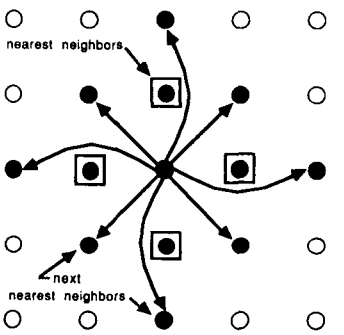

Figure 5 - Next Nearest Neighbor Connectivity

9. TABLES

Table I
Genetic Algorithm Parameters

Parameter	Value
Model	R4
Populations Size	400
Crossover Rate	0.95
Mutation Rate	0.01

Table II - Simulation Parameters

Run	Architecture	Objective Function	Training Method
1	FFQ	P1	T1
2	FFQ	P1	T1
3	FFQ	P1	T1
4	RQ	P1	T1
5	FIH	P1	T1
6	FIH	P1	T2
7	FIH	P2	T2
8	RQ	P2	T2
9	RQNNN	P2	T2

Run	# Paths Shown Per Generation	Weight Vector Encoding	Generalization Capability
1	1	LR	
2	5	LR	
3	1	Q	
4	1	LR	
5	1	LR	
6	1	LR	2nd
7	5	LR	
8	5	Q	1st
9	5	Q	3rd

Table III - Performance on Training Data Paths

Run No.	Optimal Path Length 2	3	4	5	6	7
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	-	-	-	-	-	-
4	-	-	-	-	-	-
5	-	-	-	-	-	-
6	92	92	91	90	87	87
7	11	22	17	22	20	24
8	86	84	85	87	85	87
9	42	49	51	53	55	58

Run No.	Optimal Path Length 8	9	10	11	12	13	>13
1	-	-	-	-	-	-	100
2	-	-	-	81	81	81	73
3	-	-	-	-	-	-	98
4	-	-	-	-	-	-	100
5	-	-	-	-	-	-	100
6	82	83	78	75	75	74	73
7	33	26	40	39	50	37	53
8	85	86	83	84	85	79	84
9	59	63	66	69	71	73	76

Table IV - Performance on Test Data Paths

Run No.	Optimal Path Length 2	3	4	5	6	7
1	-44	-39	-37	-38	-36	-28
2	4	-7	-5	-14	-12	-13
3	10	12	7	16	18	13
4	22	29	33	31	29	28
5	72	77	80	82	85	88
6	91	93	92	91	90	85
7	14	18	17	29	27	32
8	86	85	82	85	86	87
9	45	48	51	51	56	60
No. of Trials Averaged	32	71	79	61	98	75

Run No.	Optimal Path Length 8	9	10	11	12	13	>13
1	-33	-36	-28	-27	-27	-16	4
2	4	0	13	16	19	23	23
3	11	22	20	16	23	29	35
4	40	47	40	37	45	41	52
5	88	89	89	89	88	88	90
6	82	81	78	77	74	71	72
7	29	35	34	34	37	44	45
8	85	86	82	80	80	81	86
9	60	62	66	69	69	72	75
No. of Trials Averaged	83	44	55	32	35	19	17

Table V - Convergence Times and Total Training Set Size

Run	Generations To Converge	Training Set Size
1	21	1
2	82	5
3	25	1
4	63	1
5	73	1
6	288*	288
7	40	200
8	100	500
9	199	995

*This run was stopped before convergence; see text.

ACKNOWLEDGEMENTS

We wish to thank NASA and VERAC, Inc. for supporting this research.